Quick Sort

Introduction

Quick sort works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. For this reason, it is sometimes called partition-exchange sort. The sub-arrays are then sorted recursively [4].

Performance

In general, the performance of the sorting algorithm is analyzed from two aspects, time complexity and space complexity. Time complexity represents the time taken to execute the algorithm, which is generally considered in three cases: best case, worst case, and average case [6]. Space complexity represents the amount of memory required to complete a program [5].

Use **array** as data structure and **n** denotes the input array size, quick sort performance is as follows [4]:

Worst-case time complexity	$\boldsymbol{O(n^2)}$
Average time complexity	$\boldsymbol{O(n\ log\ n)}$
Best-case time complexity	$\boldsymbol{O(n\ log\ n)}$
Worst-case space complexity	$\boldsymbol{O(n)}$

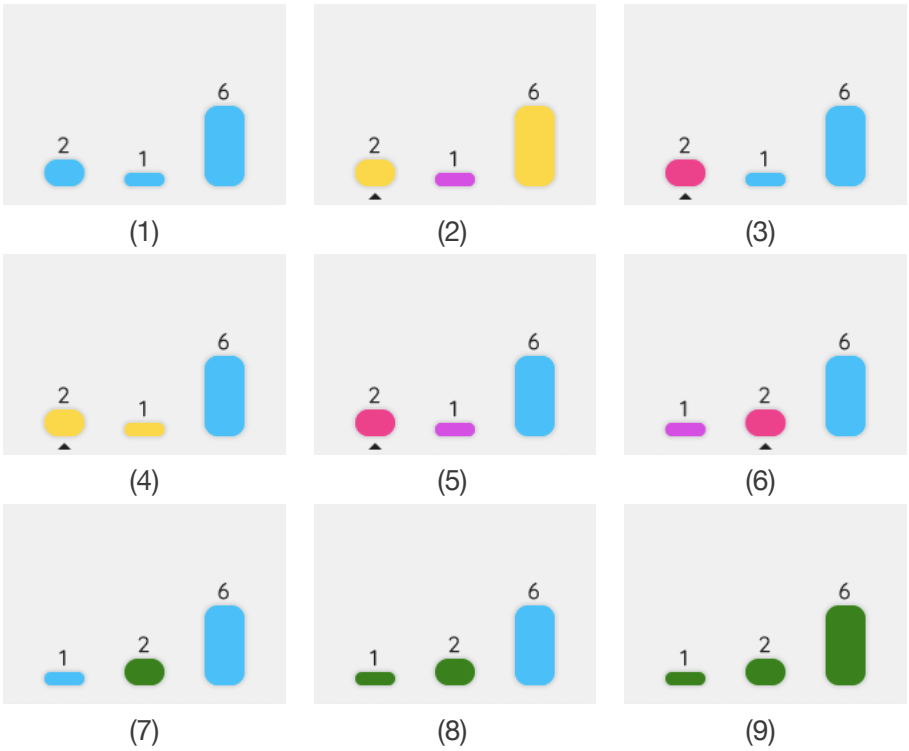
Pseudocode [3]

Algorithm: QuickSort(Arr)
Input: an array of integers Arr
Output: The result of sorting Arr

```
for each unsorted partition(a,b) do
 pivot = opsition of generate element from Arr[a] to Arr[b]
 swap(Arr[pivot],Arr[a])
 storeIndex = x + 1
   for i = x + 1 to b do
     if Arr[i] < Arr[pivot] then
       swap(Arr[i], Arr[storeIndex])
       storeIndex++
     end if
   end for
   swap(Arr[pivot], Arr[storeIndex - 1])
 end for
 return Arr
```

Example

Sorting the list: 2,1,6



Implement in programming language

Java [1]

```java
public int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1);
        for (int j=low; j<high; j++)
        {
            if (arr[j] <= pivot)
            {
                i++;

                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

public void quickSort(int arr[], int low, int high)
    {
        if (low < high)
        {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi-1);
            quickSort(arr, pi+1, high);
        }
    }
```

JavaScript [2]

```
function quickSort(arr, left, right) {
    var len = arr.length,
        partitionIndex,
        left = typeof left != 'number' ? 0 : left,
        right = typeof right != 'number' ? len - 1 : right;

    if (left < right) {
        partitionIndex = partition(arr, left, right);
        quickSort(arr, left, partitionIndex-1);
        quickSort(arr, partitionIndex+1, right);
    }
    return arr;
}

function partition(arr, left ,right) {
    var pivot = left,
        index = pivot + 1;
    for (var i = index; i <= right; i++) {
        if (arr[i] < arr[pivot]) {
            swap(arr, i, index);
            index++;
        }
    }
    swap(arr, pivot, index - 1);
    return index-1;
}

function swap(arr, i, j) {
    var temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

C [2]

```c
void swap(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}

void quick_sort_recursive(int arr[], int start, int end) {
    if (start >= end)
        return;
    int mid = arr[end];
    int left = start, right = end - 1;
    while (left < right) {
        while (arr[left] < mid && left < right)
            left++;
        while (arr[right] >= mid && left < right)
            right--;
        swap(&arr[left], &arr[right]);
    }
    if (arr[left] >= arr[end])
        swap(&arr[left], &arr[end]);
    else
        left++;
    if (left)
        quick_sort_recursive(arr, start, left - 1);
    quick_sort_recursive(arr, left + 1, end);
}

void quick_sort(int arr[], int len) {
    quick_sort_recursive(arr, 0, len - 1);
}
```

References

1. GeeksforGeeks Contributors (2018). *Java Program for QuickSort.* [Online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/java-program-for-quicksort/ [Accessed 6 Mar. 2021].
2. Runoob Contributors (2018). *Quick sort*. [Online] Runoob. Available at: https://www.runoob.com/w3cnote/quick-sort-2.html [Accessed 6 Mar. 2021].
3. Visualgo Contributors (2014). [Software] Visualgo. Available at: https://visualgo.net/en/sorting [Accessed 6 Mar. 2021].
4. Wikipedia Contributors (2021). *Quicksort*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Quicksort [Accessed 6 Mar. 2021].
5. Wikipedia Contributors (2021). *Space complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Space_complexity [Accessed 20 Mar. 2021].
6. Wikipedia Contributors (2021). *Time complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Time_complexity [Accessed 20 Mar. 2021].